Notes for the Web payment system

Andrew Payne

Story: "We are building a payment system for the Web. Our novel approach works with today's clients and doesn't let the merchants see the client's payment credentials."

TODO List:

```
- Add transaction IDs
    - merchant vouchers for hard goods
    - support for merchant reversals

- Testing framework:
    - DEJAgnu?

- Customer service Web pages:
    - on-line statements
    - change password

- Administrative pages:
    - add/delete user
    - add/delete merchant

- example of challenge/response, additional confirmation
    - allow merchant to force confirmation

- way for non-OMI users to get access to an "Open an OMI account" page

- use temporal IP address tricks to guess as sessions
    - for confirmation purposes
```

Open Problems to Address

How to handle price changes? A payment URL effectively encodes an offer to sell a product at a specified price. We need some mechansim for invalidating old payment URLs when prices are changed. One simple solution is to have "valid until" timestamps on payment URLs that specify how long the offer is good for.

How to handle double or overlapping purchases?
- transaction IDs

How to handle refunds? (user purchases access to a page, wants money back. We can refund the money but how do we deny access?)

- one idea is log scanning--scan the log for refunded accesses, and reverse refund if subsequent access is detected

```
overview.html-rev1.1        Mon Jul 18 14:40:11 1994        1
<head><title>Web Payment System Overview</title>
<h1>Web Payment System Overview</h1></head>

The basic idea is to have the pay-per-page links in the merchant's database
that point to our payment system and encode payment details (these are
called <i>payment URLs</i>). Our payment system processes the payment
order, and returns an HTTP redirect to deflect the client to the real URL
on the merchant's server.  The redirected URL is called the <i>access URL</i>.

<p>The effect for the user is a seamless link
from one merchant page to another merchant page.

<ul>
<li><A HREF="paymenturl.html">Details of payment URLs</a>
<li><A HREF="accessurl.html">Details of access URLs</a>
</ul>

<p><hr><address>Open Market, Inc.</address>
```

```
<TITLE>Encoding payment orders in URLs</TITLE>
<H1>Encoding payment orders in URLs</H1>

A payment URL looks like this:
<pre>
  http://payment.openmarket.com/pay.cgi?hash:field1=value1&amp field2=value2
</pre>
In any of the field values, the following characters are escaped:  '+',
'&', '=', ' ', and anything less than 0x20.
<p>
The URL fields encode the details of the payment order:

<dl><dt> Merchant id (<code>mid</code>)<dd>
This string uniquely identifies the merchant.</dl>

<dl><dt> Target URL (<code>url</code>)<dd>
This is the real URL (usually on the merchant's machine) where the client
will be redirected after the payment order is processed.</dl>

<dl> Amount (<code>amt</code>)<dd>
This is the amount to charge for access (presumably in US dollars).</dl>

<dl> Domain (<code>domain</code>)<dd> This is a string that identifies the
domain being granted access to on the merchant's machine. For example, a
domain might be dilbert'', which when paid for, would grant access to any
document tagged with that domain.</dl>

<dl> Signature (<code>hash</code>)<dd> The signature (hash) is computed as
the MD5 hash of {key, fieldstring}, where <code>fieldstring</code> is
everything after the colon.  The <code>key</code> is a secret key shared
between Open Market and the merchant.  The signature prevents user tampering
of the payment URL.</dl>

<p> An alternative that I initially implemented, is to compute the hash of
{mid, amt, url} and encrypt that with the merchant's secret key.  This
method has the downside of depending on encryption code, which isn't
exportable.

<H2>Creating the URLs</H2>

To aid the creation of documents with payment URLs, we will build routines
for Tcl, that let document authors write payment links as embedded Tcl
commands, like this:

<pre>
    [paylink -text "Dilbert (1.00)" -cost 1 -url http://dilbert.com \
        -domain dilbert -desc "Dilbert cartoon"]
</pre>

The markup tool takes the details of the payment, and generates a signed
URL that points to our payment server.

<h2>Open Issues</h2>

How to handle price changes?  When a merchant changes his prices (or offerings)
there might be a bunch of payment URLs cached in the network or in the user's
hands.<p>
One solution is to have a "valid until" field in the URL that specifies
how long the payment URL is good for.

<p><hr><address>Open Market, Inc.</address>
```

```
summary.html-rev1.1        Mon Jul 18 14:40:11 1994        1
<head><title>Payment System Prototype</title>
<h1>Payment System Prototype</h1></head>

These pages describe the payment system in technical detail.

<ul>
    <li><a href="overview.html">Overview</a>
    <ul>
        <li><A HREF="paymenturl.html">Details of payment URLs</a>
        <li><A HREF="accessurl.html">Details of access URLs</a>
    </ul>
    <li> HTTPD Server modifications to support payment:
        <ul>
        <li><a href="httpd-local.html">Details of local mods</a>
        <li><a href="httpd-merchant.html">For the merchant's server</a>
        <li><a href="httpd-payment.html">For OMI's payment server</a>
        </ul>

<li><a href="TODO">Andy's TODO list</a>
</ul>

<p><hr><address>Open Market, Inc.</address>
```

The problem is duplicate payments. "reloads" or double clicks can cause
a customer to be charged several times for a single item.

Hard Goods

In this case, we should have merchant generated transaction IDs. (These
are necessary, because the merchant needs to match up the hard goods order
with the payment information that comes back from Open Market).

With these IDs, we can suppress duplicate charges by keeping a table indexed
by ID. We only settle each ID once.

Information Goods

The important difference between hard goods and information goods is that
purchasing the same information item more than once doesn't make any
sense.

In the current system, the {merchant id, ticket} uniquely identify what was
purchased, and {expire} specified how long the purchase was good for.
The {merchant id, ticket} is used as a transaction ID to filter out duplicate
purchases.

Strawman Proposal

In the transaction table, maintain two fields (in addition to whatever
fields are needed for the rest of the transaction information):

- transaction ID
  - for hard goods, supplied by the merchant
  - for info goods, use {merchant id, ticket}

- expire time
  - supplied by the merchant for information goods
  - default to some reasonable time for hard goods (one month)

For each transaction, we consult this database for duplicate checks. If the
transaction already exists, we forward a message to the effect (and don't
charge for the duplicate). Otherwise, we enter a new transaction. Also,
we expire old transactions out of the database.

```
<TITLE>HTTPD Modifications, Local Details</TITLE>
<H1>HTTPD Modifications, Local Details</H1>

We've made several modifications to NCSA's HTTPD 1.3 server to support
payment.

<p>
The sources for OMI's modified server are in:
<pre>
    /omi/payment/web/httpd_1.3
</pre>
Our sources include some modifications not described in this documentation,
including conversion to use GNU Autoconf, and fixing many compiler warnings.<p>

On the Sun, these sources are linked (and built) in:
<pre>
    /omi/solaris/payment/web/httpd_1.3
</pre>

<p><hr><address>Open Market, Inc.</address>
```